VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza SQL SELECT příkazů z webu StackOverflow

Analysis of StackOverflow SQL Select Command

2020 Daniel Hloušek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Daniel Hloušek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Analýza SQL SELECT příkazů z webu StackOverflow
Analysis of StackOverflow SQL Select Command

Jazyk vypracování: čeština

Zásady pro vypracování:

StackOverflow představuje jednu z největších Q&A služeb na světě. Jednou z kategorií jsou otázky zaměřené na problémy spojené s psaním SQL kódu. StackOverflow databáze je volně k dispozici ke stažení. Cílem práce je vytvořit databázi a nástroj, který umožní provést analýzů příkazů SQL Select, které se nacházejí v otázkách a odpovědích.

Práce bude probíhat v následujících krocích:
1. Vytvoření databáze, kde se budou nacházet již zpracované příkazy SQL Select. Zpracované SQL příkazy budou uloženy ve formátu XML, kde jednotlivé XML tagy budou odpovídat uzlům derivačního stromu vzniklého při parsrování SQL příkazů.
2. Vytvoření nástroje, který umožní nad SQL dotazy, uloženými ve formátu XML, provést XQuery dotazy.
3. Vytvoření několika vzorových XQuery dotazů, které naleznou několik typů SQL dotazů:
a) Používající zastaralého zápisu spojení tabulek.
b) Obsahující nezávislý poddotaz v EXISTS.
c) Obsahující tabulku, která není propojena, žádným spojením s ostatními.
d) Redundantní konstrukce distinct (kupříkladu v poddotazu IN).
4. Testování výsledného řešení

Výsledný systém bude mít následující funkce:
1. Při analýze jednotlivých otázek bude docházek k parsrování SQL příkazů pro databázové systémy SQL Server a PostgreSQL.
2. Systém uloží jednotlivé otázky do vlastní databáze, kde budou uloženy i výsledky analýzy.
3. V interní databázi bude uložen derivační strom jednotlivých SQL příkazů a aplikace bude implementovat jednoduché vyhledávání v těchto syntaktických stromech.

Seznam doporučené odborné literatury:

[1] Melton, Jim, and Alan R. Simon. SQL: 1999: understanding relational language components. Elsevier, 2001.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 19. dubna 2020 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 19. dubna 2020 .....

Rád bych na tomto místě poděkoval panu Doc. Ing. Radimu Bačovi, Ph.D., který mi s prací pomáhal, ať už osobně, tak i prostřednictvím video hovorů v nelehkém období pandemie.

Abstrakt

StackOverflow je velmi populární Question&Answer služba, přičemž jedno z mnoha probíraných témat, které se na StackOveflow řeší jsou SQL SELECT dotazy. Cílem této práce je vytvořit databázi a nástroj, který umožní provádět analýzu SQL příkazů získaných a zpracovaných právě ze StackOverflow. Databáze bude obsahovat derivační stromy SELECT dotazů v XML formátu. Aplikace umožní spouštět XQuery dotazy na databázi. Navíc budou pro demonstraci připraveny čtyři XQuery dotazy.

Klíčová slova: StackOverflow; SQL; SELECT; xQuery; analýza

Abstract

StackOverflow is popular Question&Answer service. One of the main topics are SQL SELECT queries. Goal of this bachelor thesis is to create database and application, that provides analysis of StackOverflow SQL queries stored in derivation trees in database as XML format. Application can be used to run own XQuery queries and also there are four predefined xQuery

Keywords: StackOverflow; SQL; SELECT; xQuery; analysis

Obsah

Seznam použitých zkratek a symbolů

GSP	–	General SQL Parser
SQL	–	Structured query language
XML	–	Extensible Markup Language
API	–	Application programming interface
PL/SQL	–	Procedural Language for SQL
FLWOR	–	For, Let, Where, Order by, Return
GUI	–	graphical user interface
CLI	–	Command-line interface
SO	–	StackOverflow
PC	–	Personal Computer
MVC	–	Model View Controller

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

StackOverflow je dlouhodobě jedna z nejpoužívanějších Question&Answer služeb. V informatice je běžné hledat postupy různých řešení na internetu a právě StackOverflow má obrovskou databázi řešení. Jednou z kategorií, které se na StackOverflow probírají jsou SQL, konkrétně SELECT, dotazy.

Cílem této práce je vytvořit dvě aplikace. První aplikace zpracuje data ze StackOverflow a vytvoří BaseX databázi. Databáze bude obsahovat získané SELECT dotazy ze StackOverflow převedené na derivační stromy ve formátu XML. K převedení SQL dotazů na derivační stromy v podobě XML je použit GSP - General SQL Parser, vyvinut společností Gudusoft.

Druhá aplikace je webová aplikace napsaná v C# s použitím ASP.NET. Aplikace komunikuje s vytvořenou databází. Webová aplikace umožní psaní vlastních XQuery dotazů, jejichž výsledky mohou být použity pro analýzy uložených derivačních stromů SQL dotazů. K dispozici také budou čtyři předdefinované XQuery dotazy, které demonstrují funkcionalitu a možnosti XQuery.

V uložených datech můžeme rozpoznat, zda je SQL dotaz na webu StackOverflow vložen tázajícím se uživatelem na nějakou problematiku, nebo naopak někým, kdo takovýmto SQL dotazem radí ostatním uživatelům. Můžeme tedy například zjistit, kolik odpovědí obsahuje chybný SQL příkaz, jaké oblasti SQL trápí uživatele nejvíce, kolik dotazů se opakuje a má pouze jiné názvy sloupců, nebo můžeme třeba zjistit, jaké typy dotazů jsou nejdiskutovanější.

V první kapitole se věnuji teoretické části. Jsou zde popsány jednotlivé technologie, se kterými jsem pracoval. Je zde také popsán derivační strom. Dále je zde podkapitola věnující se výběru databázového úložiště pro tuto práci.

V druhé kapitole se věnuji popisu StackOverflow Analyzer aplikace. Nejdříve je popsána konzolová část, včetně všech tříd a metod. Dále je zde podkapitola o webové části aplikace, kde jsou popsané jednotlivé části.

Ve třetí kapitole se věnuji testování na vlastních datech. Rozebírá se zde měření jednotlivých částí konzolové části aplikace. Měření se týká procesů od zpracování vstupního souboru, po vytvoření databáze.

V Závěru zhodnocuji výsledky této práce, přínosy a dosažené cíle.

2 Teoretická část

Teoritecká část se zabývá použitými technologiemi, a to SQL parserem, rešerší databázových systémů, a v neposlední řadě technologii XQuery.

2.1 SQL parser a derivační strom

Data stažená ze StackOverflow obsahují SQL dotazy v textové podobě. SQL dotazy se převádějí na derivační stromy, jelikož to umožní komplexní analýzy popisované v úvodu. Je tudíž potřeba tyto dotazy převést do XML a až pak uložit do databáze. Knihovna gsqlparser právě takové parsrování nabízí.

2.1.1 General SQL Parser

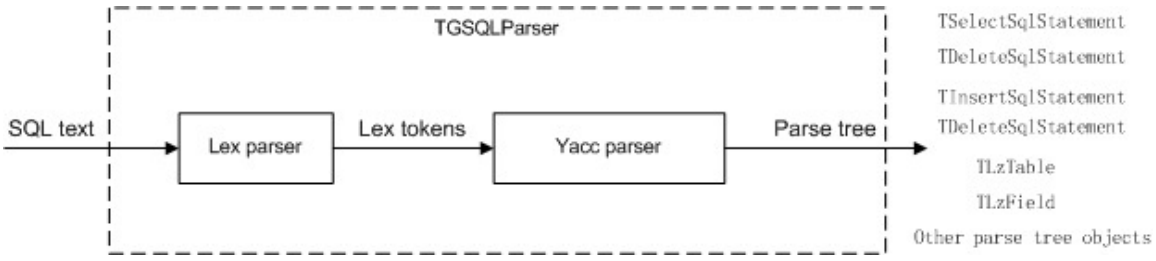
General SQL Parser, dále jen GSP, je nástroj pro práci s SQL. Umožňuje parsrování, formátování, modifikaci a analýzu SQL dotazů. Vyvinula jej společnost GuduSoft a GSP lze získat v mnoha variacích. Pro tuto práci, která je psána v C#, je použit General SQL Parser v3.3.1.0 pro .NET. Co se týče licencí, tak pro tuto práci je knihovna použita v Trial verzi, což má lehké omezení - GSP zpracuje pouze dotazy kratší než 10000 znaků. A druhé omezení - trial verze vyprší za 90 dní od data stažení. Nicméně i přes tato omezení se GSP jeví jako nejlepší volba pro tuto práci, neboť obsahuje API pro C# a má naimplementovanou vlastní gramatiku. Knihovnu lze tedy stáhnout a používat v C# aplikaci. Po připojení knihovny do aplikace máme k dispozici třídu TGSqlParser, přes kterou bude probíhat veškerá parsrovací logika. Dále nám také přibude entita EDbVendor, která nám určuje databázový engine. Právě EDbVendor se předává do konstruktoru TGSqlParser a pomáhá třídě pak určit, kterou gramatiku použít, protože pro každý databázový engine se gramatika o něco liší. [1]

EDbVendor se ve verzi pro C# může nastavit na 14 různých enginů. Mezi podporované patří MySQL, Oracle, nebo třeba SQL Server. [2]

Jak je vidět, GSP je připraven na mnoho databázových enginů - a to je výpis enginů, které GSP podporuje jen pro C#, takže kdybychom zahrnuli i třeba GSP verzi pro Javu, tak je seznam ještě mnohem rozsáhlejší.

2.1.2 Derivační strom

Jedna z předností GSP je právě parsrování SQL dotazů na derivační stromy v XML formátu s předpřipravenou gramatikou. Derivační strom v podstatě představuje syntaktickou strukturu SQL řetězce podle formální gramatiky. Tato gramatika může být jak jednoznačná, tak víceznačná. Víceznačná gramatika umožňuje větvení při parsrování - vznikne tak více výstupů, což u této práce není žádoucí. Gramatika GSP je jednoznačná - pro jeden SQL dotaz vrátí vždy jeden a ten samý derivační strom. Derivaci samotnou můžeme provádět na jakémkoliv slově, větě, nebo nad jakoukoliv textovou reprezentací. Formální gramatika G je dána čtveřicí (N, T, P, S), kde



Obrázek 1: TGSQLParser - Workflow [6]

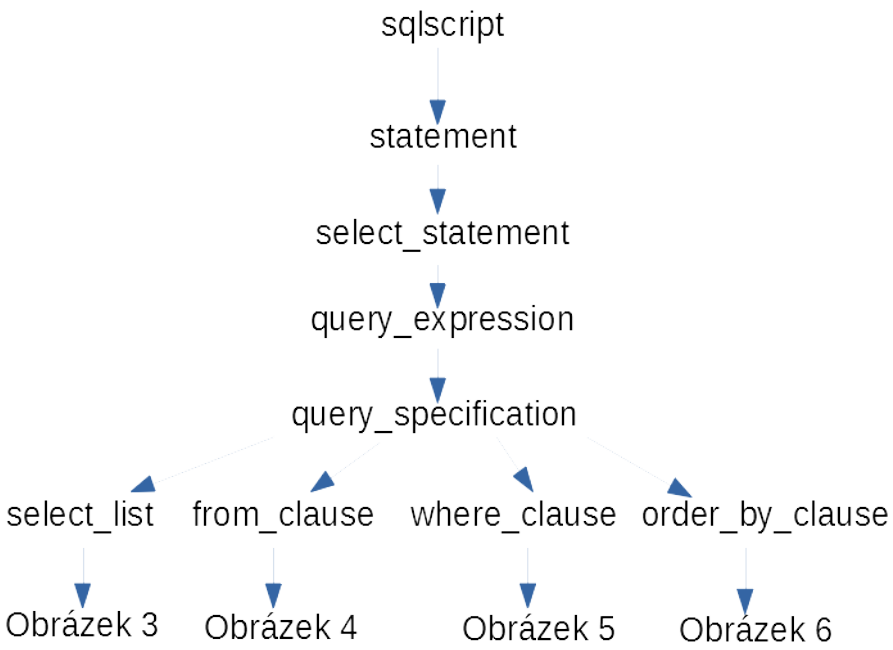
N je konečná množina neterminálních symbolů, T je konečná množina terminálních symbolů, P je konečná množina odvozovacích pravidel a S je počáteční symbol. Taková gramatika zpravidla generuje slova, přičemž podmnožina všech slov dané gramatiky se nazývá formální jazyk. V derivačním stromě pak platí, že terminály v gramatice jsou listy, neterminály jsou uzly, které mají potomka a počáteční symbol je kořen stromu. [3] [4] Výhoda GSP je v tom, že má již vlastní gramatiku a není zde potřeba vytvářet novou. Vytvoření gramatiky není triviální proces a dala by se jí pokrýt i celá jiná bakalářská práce. Jak vlastně takový překlad probíhá? Nejdříve SQL dotaz putuje do lexeru. Lexer vytvoří list, kde každá část SQL dotazu dostane určitý příznak, neboli token. Tento list tokenů se pak předá parseru jako vstupní parametr. Tento parser zkusí na základě gramatických pravidel zkonstruovat derivační strom. Derivační strom se uloží do paměti a následně je možné jej vyexportovat do XML. Pokud se při vytváření stromů narazí na syntaktickou chybu, což znamená, že slovo (zpracovávaný SQL dotaz) do gramatiky nepatří, tak se strom nevytvoří. Dále se strom naformátuje podle toho, zda se jedná o Select, Insert, Delete, nebo Update. Každá SQL operace má tedy ve výsledku jinou strukturu derivačního stromu. Viz obrázek 1 [5]

Jak přesně vypadá dekódování SQL gramatiky na SELECT příkazu je vidět ve výpisu 1.

```
SELECT e.last_name    AS name,
       e.salary * 12  "Annual Salary"
FROM   scott.employees AS e
WHERE  e.salary > 1000
ORDER  BY
  e.first_name,
  e.last_name;
```

Výpis 1: Jednoduchý SQL SELECT

Takovýto SQL dotaz po parsrování s použitím GSP gramatiky můžeme vidět na obrázcích 2 - 6



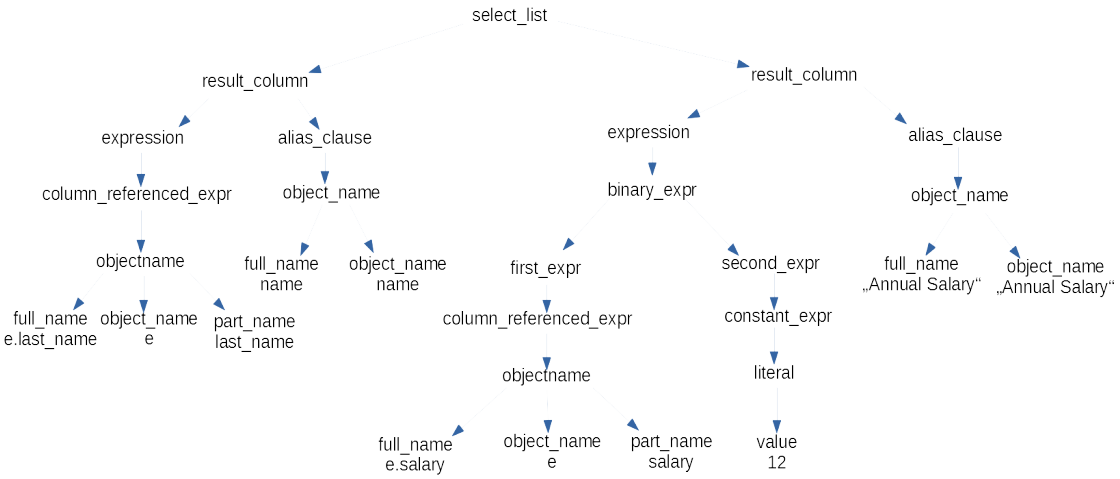
Obrázek 2: TGSQLParser - část 1

Takto zpracovaný SQL příkaz je dále zpracován a uložen do XML souboru. Pozor, jedná se o gramatiku použitou při SELECT dotazu. Pro ostatní typy dotazů je výstup jiný, ovšem v této práci se zajímáme výhradně o SELECT dotazy. Nutno však podotknout, že pomocí GSP můžeme parsrovat i PL/SQL. [7]

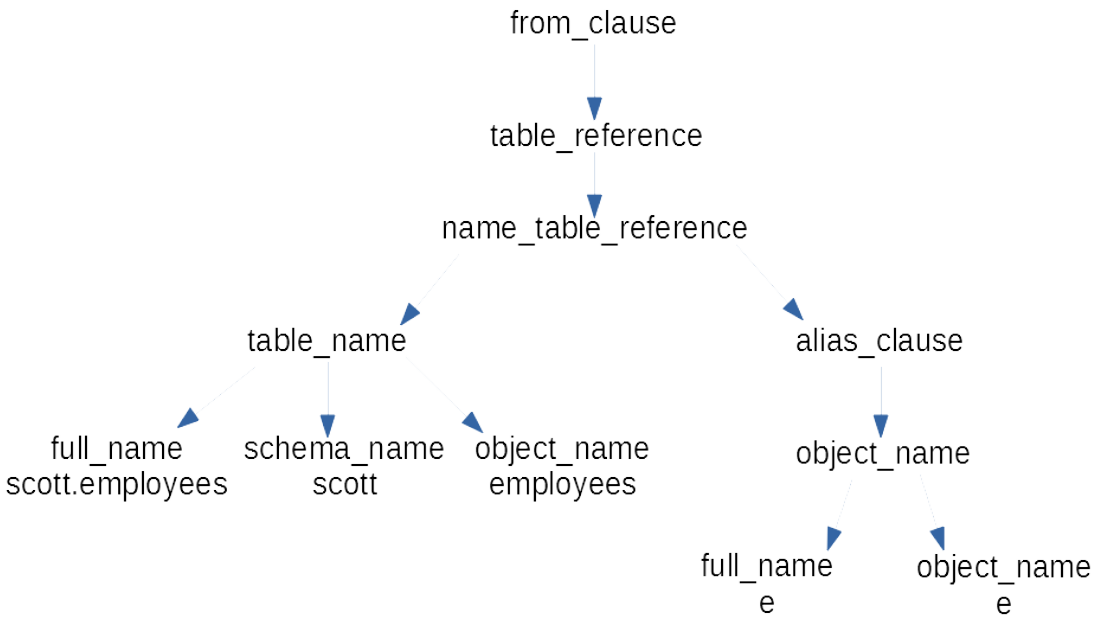
2.2 Rešerše Databázových systémů

Před samotnou realizací aplikace bylo důležité zvolit vhodné datové úložiště. Při výběru se zvažovaly technologie SQL Server, MySql a BaseX. Důležité aspekty pro výběr vhodné databáze jsou v mém případě, kromě rychlosti, také pohodlnost spouštění XQuery dotazů, způsob práce s XML soubory a integrace do C#. XPath a XQuery technologie jsou popsány v další kapitole.

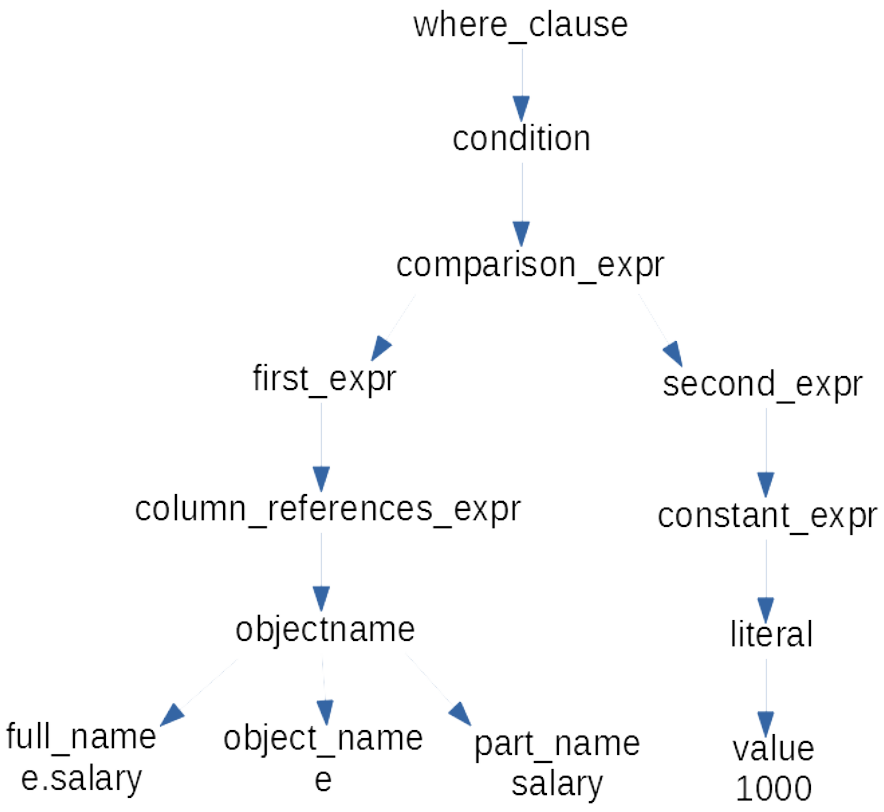
Pro výběr úložiště mi pomohla tabulka níže. Aspekt „Pohodlnost“ se hodnotí na stupnici od 1 do 5 (1 – nejméně pohodlné, 5 – nejvíce pohodlné), podle toho, jak přehledně a pohodlně se pracuje s XML daty a XQuery dotazy na danou databázi. Zdůvodnění k těmto hodnotám jsou k nalezení níže pod tabulkou. Rychlost byla měřena spuštěním stejného dotazu postupně na všechna úložiště, obsahující 40 000 záznamů. Takto bude aplikace totiž fungovat - nad enormním počtem XML dokumentů provede zvolený XQuery dotaz.



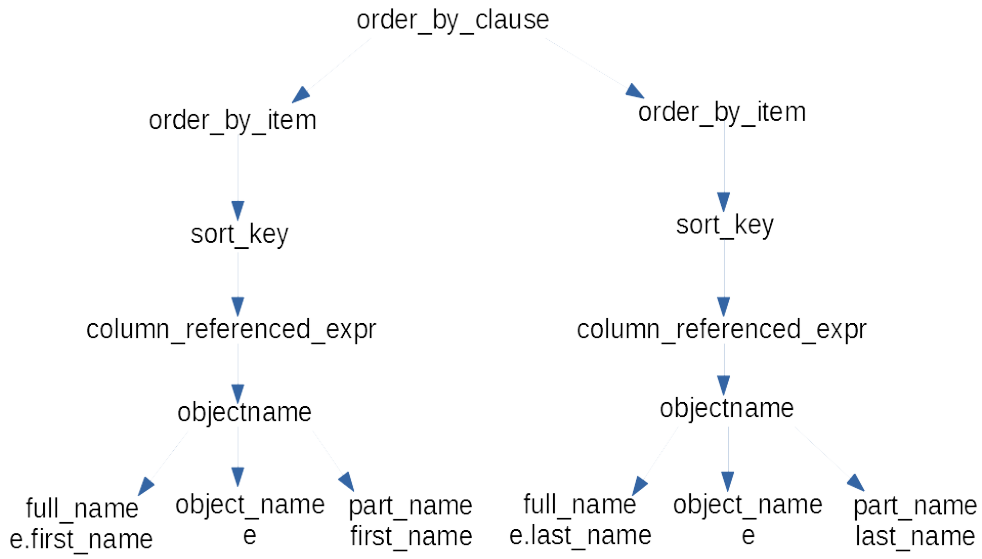
Obrázek 3: TGSQLParser - část 2



Obrázek 4: TGSQLParser - část 3



Obrázek 5: TGSQLParser - část 4



Obrázek 6: TGSQLParser - část 5

Úložiště	Rychlost (s)	Pohodlnost
MySQL	0,01	2
SQL Server	1	3
BaseX	2,79	5

Tabulka 1: Rešerše databázových systémů

2.2.1 MySQL

MySql bohužel nepodporuje XQuery, nabízí však funkce pro základní práci s XML. Disponuje například funkcí ExtractValue(string xml, string xpath). Funkce má 2 parametry – XML string a XPath string. Jelikož v mé databázi reprezentuje jeden řádek jedno xml, tak jsem ExtractValue umístil do WHERE klauzule. XPath jsem obalil do count() funkce, takže pokud XPath požadovaný uzel našel, tak count() vracel nenulový výsledek. Jak jsem již zmínil, MySql nepodporuje přímo XQuery, takže pětici FLWOR funkcionalit zde musíme nahradit technologií XPath a samotnou syntaxí MySql procedur. Z tohoto důvodu MySql není vhodným databázovým enginem pro tuto práci.

Ukázka kódu

```
-- Vrací id záznamů, kde je zastaralý způsob spojení tabulek

SELECT id FROM queries
```

```
WHERE ExtractValue(xml_data,'count(/sqlscript/statement/select_statement/
    query_expression/query_specification/from_clause/table_reference[@type="
    objectname"])') != 0;
```

Výpis 2: Ukázka MySQL a xQuery

2.2.2 SQL Server

SQL Server nám nabízí jako datový typ pro XML záznamy přímo datový typ sloupce - XML. Takže z datového pohledu je SQL Server připravenější, než MySql. Používáme zde funkci query, kterou voláme přes „.“ přímo na XML sloupec v dotazu. Do této funkce píšeme XQuery dotaz, který může obsahovat i klasickou FLWOR konstrukci. Na výsledek query funkce můžeme zavolat funkci value(xQuery, SQL datový typ), která nám extrahuje výsledek tak, abychom s ním dále v dotazu mohli pracovat. SQL server této práci z pohledu samotného XQuery tedy vyhovuje. I přes to, že SQL Server podporuje XQuery, tak neumožní vstupní XML, nebo výsledky (pokud je výsledek XML) přehledně zobrazit v nějakém grafu. Takže k přehlednému zobrazení pro testování, zda je XQuery dotaz napsán správně, je potřeba použít jiný externí program – a to je důvod, proč SQLServer není databázovým enginem této práce.

Ukázka kódu

```
--Vrací id záznamů, kde je zastaralý způsob spojení tabulek
WITH XMLNAMESPACES(N'http://www.sqlparser.com/xml/sqlschema/1.0' as x )
SELECT id
FROM queries
WHERE xml_data.query('
/x:sqlscript/x:statement/x:select_statement/x:query_expression/x:
    query_specification/x:from_clause/x:table_reference[@type="objectname"]
').value('.','VARCHAR(255)') NOT LIKE '';
```

Výpis 3: Ukázka SQL Server a xQuery

2.2.3 BaseX

BaseX je databázový engine navržený přímo pro práci s XML. BaseX je open source a podporuje jak desktopové GUI, tak CLI. Mimo jiné nabízí BaseX API pro práci s C#. Za účelem zjištění, zda je BaseX vhodný nástroj pro tuto práci, jsem použil desktopové GUI. Do programu se nahraje XML soubor, odpovídající DTD schématu, nad kterým se pak provádí XQuery dotazy. Abych mohl XQuery provést na více XML souborech, tak jsem obsah jednotlivých XML sloučil do jednoho souboru a obalil tagem DB. V BaseX se DB rozumí jeden XML (dtd) soubor. Co se týče XQuery, tak to po spuštění vrací jednotlivé uzly z XML. Největší výhoda BaseX je v tom, jak přehledně v různých grafech se dá XML soubory procházet.

Ukázka kódu

```
--Vrací uzly záznamů, kde je zastaralý způsob spojení tabulek
declare default element namespace "http://www.sqlparser.com/xml/sqlschema/1.0";
for $xml in doc("testRychlosti.dtd")
/sqlscript/statement/select_statement/query_expression/query_specification/
    from_clause/table_reference[@type="objectname"]
Return $xml
```

Výpis 4: Ukázka BaseX a xQuery

2.2.4 Shrnutí

Přesto, že je BaseX oproti MySql a SQL Serveru pomalejší, tak je práce s ním daleko přehlednější a příjemnější. Proto BaseX použiji jako hlavní databázový engine pro tuto práci.

2.3 Úvod do XQuery

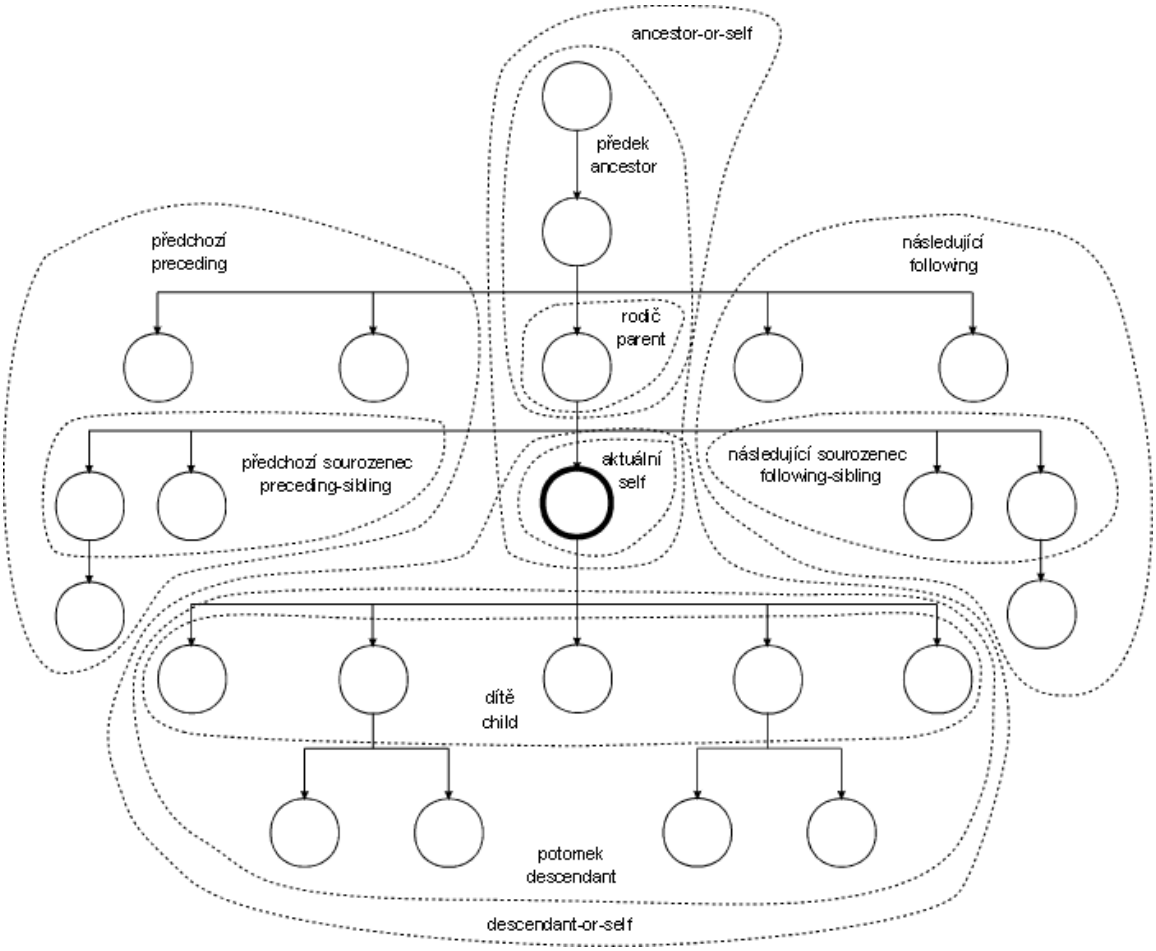
XQuery je standardizovaný jazyk pro dotazování semistrukturovaných dokumentů, databází, webových stránek a v podstatě čehokoliv, co má semistrukturovanou strukturu. XQuery je pod správou W3C a momentální nejnovější verze je 3.1. XQuery je velmi rozšířené a multiplatformní. Kromě získání dat z databází, XML souborů, nebo třeba webových dokumentů, lze XQuery použít dokonce i z CGI skriptů. Dokáže vracet výsledky v XML, které mohou být dále zpracovávány. V podstatě se dá říct, že XQuery je něco jako SQL pro databáze, nejedná se však o SQL. Aby se XQuery dalo použít přímo na databázi, daný databázový engine musí podporovat XQuery. Většina nejvýznamnějších databázových enginů XQuery dnes již podporuje. V této práci se však XQuery používá pro XML dokumenty. XQuery má obrovský potenciál v analytice, kdy dokáže generovat Enterprise reporty a nejrůznější zprávy. Samotné XQuery je navrženo s důrazem na optimalizaci a pre-kompilaci, což z něj dělá vhodný deklarativní jazyk i v případě enormních XML dokumentů a stromově orientovaných databází, jako třeba BaseX.

Co tedy XQuery dokáže se dá přirovnat k SQL, ale s tím, že se aplikuje na XML soubor. XQuery vychází z technologie XPath, a kdo ovládá XPath, ovládá v podstatě nejzákladnější použití XQuery. [8]

2.3.1 XPath

XPath je používán k navigaci mezi elementy a atributy v XML souborech, včetně webových HTML stránek. Jedná se o syntaxi, díky které jsme schopni identifikovat jeden nebo více uzlů v XML dokumentu. Syntaxe není jen o psaní cesty od uzlu k uzlu, ale nabízí k použití přes 200 funkcí. Jedná se o hlavní prvek v XSLT standardu, a stejně jako XQuery je pod správou W3C organizace. První verze XPathu byla vydaná už v roce 1999. Momentálně je XPath ve verzi 3.0 z roku 2014.

XPath pracuje se sedmi typy uzlů: element, atribut, text, namespace, instrukce, komentář a uzly dokumentu. XPath uvažuje stromovou reprezentací XML. Uzly mohou mít různé vztahy, neboli osy, popisující pohyb v dokumentu. Přehled všech vztahů naleznete na obrázku 7. [9]



Obrázek 7: XPath - osy [10]

Vazby jsou vyhodnocovány vzhledem ke kontextovému uzlu, což je právě zpracovávaný uzel. Ostatní uzly, které se zrovna nevyhodnocují, ale k jejich vyhodnocení dojde, tvoří množinu kontextových uzlů. Těchto vazeb se využívá právě při tvoření XPathu. Nedílnou součástí XPathu jsou Location steps, přičemž každý XPath musí mít jeden, nebo více location steps, a ty jsou odděleny lomítkem. Na následujícím příkladu vysvětlím, co přesně je myšleno pojmy Location step, kontextový uzel a množina kontextových uzlů. Model pro příklady XPathu je následující:

Příklad 2.1 *<tvary>*
<obdélník>
<délka jednotka=”m”>2</metr>
<šířka jednotka=”m”>1</šířka>
<popis>Obdélník nemá všechny strany stejně dlouhé<popis>

</obdélník>
<čtverec>
<délka jednotka=”cm”> 20</metr>
<šířka jednotka=”cm”> 20</šířka>
<popis>Čtverec má všechny strany stejně dlouhé<popis>
<čtverec>
</tvary>

Můžeme například chtít najít všechny tvary, jejichž přímý potomek "délka"má atribut "jednotka", a ten se rovná "m", jako metr. XPath by pak vypadal takto.

Příklad 2.2 */tvary/obdélník/délka[@jednotka="m"]/parent::obdélník*

Na příkladu 2.2 je v prvním kroku zpracování kontextový uzel "tvary". Množina kontextových uzlů jsou "obdélník"a "čtverec"i s jejich podmnožinou tagů. Location step je trojice: axis, uzel, predikát. Location step je zapsán ve formátu axis:uzel[predikát]. Povinná část jednoho location step je uzel. V příkladu můžeme vidět 4 locatiion steps. Tvary, obdélník, délka jsou uzly. Parent značí axis. A výraz @jednotka="m"je predikát. Pokud jsou v XPathu dvě lomítka za sebou, je to zkrácený zápis pro axis descendent-or-self. Co se týče vztahů, tak ty vysvětluje obrázek 7, s tím, že kontextový uzel je v obrázku zaznačen pod pojmem aktuální, nebo self.

2.3.2 Základní struktura XQuery

XQuery v podstatě rozšiřuje XPath o další funkcionalitu. Jednou z nejzákladnějších funkcí v XQuery je funkce doc(), jejíž parametr je soubor, na kterém dotaz vykonáváme. Pro demonstraci XQuery použiji následující XML strukturu z pomyslného souboru "geometrické-tvary.xml":

Příklad 2.3 *<tvary>*
<obdélník>
<délka jednotka=”m”>2</metr>
<šířka jednotka=”m”>1</šířka>
<popis>Obdélník nemá všechny strany stejně dlouhé<popis>
</obdélník>
<čtverec>
<délka jednotka=”cm”> 20</metr>
<šířka jednotka=”cm”> 20</šířka>
<popis>Čtverec má všechny strany stejně dlouhé<popis>
<čtverec>
</tvary>

XQuery dotaz by vypadal takto:

Příklad 2.4 *doc("geometrické-tvary.xml")//obdelnik[1]*

Jak je vidět, na takto jednoduchém příkladě se skutečně jedná pouze o XPath s jakýmsi prefixem. Jednou z hlavních funkcionalit XQuery je FLWOR. FLWOR je akronym pro for, let, where, order by, return. FLWOR je tedy akronym pro řadu klíčových slov, díky kterým XQuery umožňuje cyklicky procházet uzly, přiřazovat uzly do proměnných, filtrovat uzly, řadit uzly a vracet uzly. Zde je další podobnost s SQL. Jednotlivé FLWOR příkazy tvoří konstrukci XQuery, která je podobná příkazům SQL. V SQL máme například SELECT, FROM, WHERE, ORDER BY a v XQuery například XPATH K UZLU, DOC(), WHERE, ORDER BY, RETURN. Je patrné, že v XQuery se stejně jako u SQL nemusí využívat celá maximální funkcionalita, takže můžeme použít třeba jen FOR a RETURN, stejně jako SELECT a FROM. WHERE je zde nazýváno stejně jako v SQL a podmínky se vyhodnocují porovnáváním atributů. [11] Co se týče vracení hodnot, tak zde můžeme použít užitečnou funkci data(), která extrahuje data z paremetru, což je atribut zvoleného uzlu. XQuery umožňuje vracet několik hodnot, proto je výhodně použít jej v HTML. Můžeme mít například ul tagy a mezi nimi XQuery, který bude z XML souboru generovat li položky. [12]

XQuery, stejně jako SQL, nabízí mnoho funkcí. Jelikož XQuery pracuje s technologií XPath, tak verze XQuery 1.0 nabízí naprosto stejné funkce jako XPath 2.0. Funkce se volají tam, kde je psán XPath.

Následující příklady využívají XML z příkladu 2.3.

```
--pro každý tag "popis", jehož rodič je jakýkoliv tag a předek je "tvary", nastaví textovou
    hodnotu velkým fontem.
for xindoc("geometrické − tvary.xml")/tvary/ ∗ /popis/text()returnupper − case(x)
```

Výpis 5: Volání funkcí v xQuery

Dále se dají funkce volat při práci s atributy.

```
--textový obsah tagů "popis", jejichž rodič je jakýkoliv tag a předek je tag "tvary", vyří
    zne znaky od pozice 1, do pozice 5 a tento výřez vrátí.
for $x in doc("geometrické−tvary.xml")/tvary/∗/popis/text()
return substring($x,1,5)
```

Výpis 6: Volání funkcí v xQuery

Poslední možnost, kde lze v XQuery použít funkce je v klauzuli let, při definici proměnné.

```
--nastaví obsah proměnné $popis malým písmem a vrátí jej
for $popis in doc("geometrické-tvary.xml")/tvary/*/popis/text()
let $lowerPopis := (lpwer-case($popis))
return $lowerPopis
```

Výpis 7: Volání funkcí v xQuery

3 Popis StackOverflow Analyzer aplikace

StackOverflow Analyzer je aplikace, nebo spíše soubor dvou aplikací, které jsou součástí vypracování této bakalářské práce. Aplikace StackOverflow Analyzer - konzolová část slouží ke zpracování vstupního SO XML souboru staženého ze StackOverflow a vytvoření BaseX databáze. Spouští se pomocí příkazové řádky a má dva parametry. První parametr je cesta k SO XML dokumentu. Druhý parametr je nepovinný, pokud jej ale zadáme s hodnotou "true", tak se kromě zpracování XML databáze na lokálním serveru přímo vytvoří. V takovém případě se předpokládá, že je BaseX server spuštěn a hodnoty pro připojení jsou na lokálním serveru defaultní. To znamená, že jméno a heslo je "admin"a port je 1984. Co tedy StackOverflow Analyzer - konzolová část přesně dělá, je to, že aplikace postupně zpracuje vstupní SO XML soubor, vytvoří derivační stromy a vytvoří BaseX databázi. Aplikace je psána v C# s využitím .NET Core 3.1. Druhou aplikací je aplikace webová, která slouží k psaní XQuery dotazů. Webová aplikace je psána taktéž v C# a využívá ASP.NET MVC model. Nejdříve se ale pojďme podívat na konzolovou část aplikace.

3.1 StackOverflow Analyzer - konzolová aplikace

Konzolová část StackOverflow aplikace slouží primárně ke zpracování vstupního SO XML dokumentu, jehož výsledek je XML, které slouží k vytvoření BaseX databáze. Popřípadě aplikace tuto databázi může i vytvořit. Spouští se pomocí příkazové řádky a jako první parametr je cesta k SO XML souboru. Druhý parametr je nepovinný, pokud je true, dojde k vytvoření BaseX databáze na lokálním serveru. Předpokládají se defaultní hodnoty pro připojení k databázi. Spustit se tedy dá konkrétně příkazem:

```
StackOverflow_Analyzer Posts.xml
```

Výpis 8: Spuštění aplikace

SO XML soubor lze stáhnout na této adrese: `https://archive.org/details/stackexchange`. Jedná se o soubor "stackoverflow.com-Posts.7z". BaseX databázi je možno stáhnout zde: `http://basex.org/download/`. Nyní se pojďmě podrobně podívat na to, co se děje uvnitř aplikace StackOverflow - konzolová část. Aplikace si vytvoří složku files, ve které jsou souboru, které v průběhu zpracovávání SO XML souboru vznikají. Výsledná složka files s vygenerovanými soubory má následující strukturu:

- databases
  - db.xml - jedná se o XML, sloužící k vytvoření BaseX databáze - viz kapitola 3.1.1.3
- trees

  - treeX.txt - jedná se o SQL dotaz, přičemž X v názvu souboru je pořadové číslo zpracování. Každý treeX.txt má přiřazený treeX.xml se stejným číslem X, který obsahuje derivační strom daného dotazu. - viz kapitola 3.1.1.2
  - treeX.xml - jedná se o derivační strom, kde X je pořadové číslo zpracování. Každý treeX.xml má k sobě přiřazený treeX.txt se stejným pořadovým číslem X, jehož obsah je SQL dotaz. - viz kapitola 3.1.1.2

- codes.txt - obsahem tohoto souboru jsou hodnoty, mezi tagy <code></code> v SO XML souboru. Soubor vzniká spuštěním funkce GetCodeAttr, která slouží jen pro debug. - viz kapitola 3.1.1.1

- jsonRows.json - obsahuje - soubor, který vzniká během procesu zpracování SO XML souboru. Vytváří jej metoda GenerateMsSqlCodeAttr. - viz kapitola 3.1.1.1

- XmlStructure.xml - soubor vznikající v metodě GenerateMyXml. Jedná se o pomocný soubor, který obsahuje určený počet řádků ze souboru SO XML souboru. - viz kapitola 3.1.1.1

3.1.1 Struktura aplikace a popis vlastních tříd

Na třídním diagramu - obrázek 8 je znázorněna struktura a závislost implementovaných tříd.

Z třídního diagramu 8 je vidět, že proces zpracování XML probíhá sekvenčně a každá třída řeší vlastní kus problematiky. Toto je také důvod, proč není model nějak více provázaný.

Pro přehlednost, jak vše probíhá, jsem vytvořil sekvenční diagram - obrázek 9.

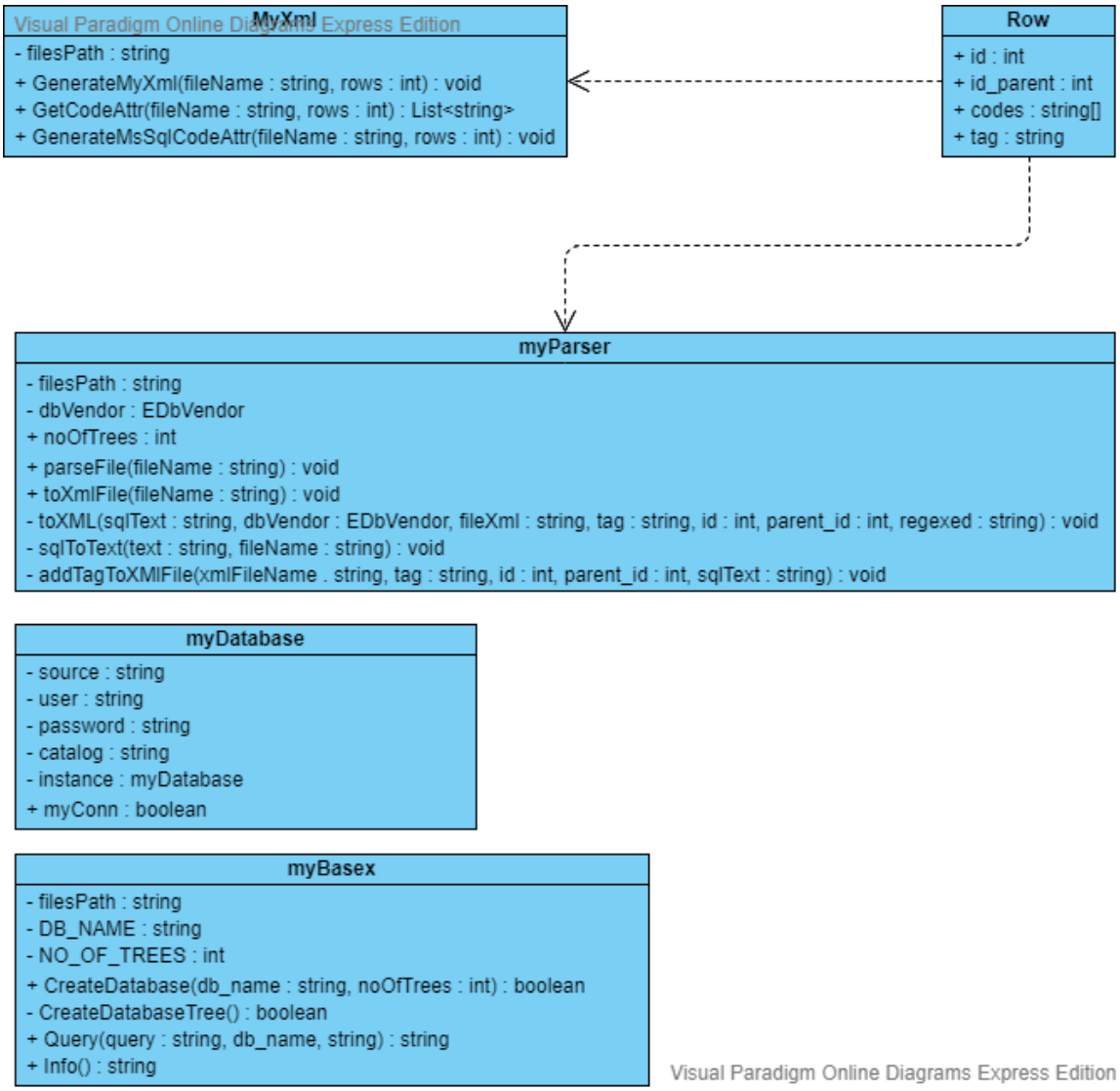
Pro přehlednost toho, jak kolují soubory systémem, přikládám DFD diagram - obrázek 10

Následuje pseudokód pro metodu main. Pseudokódy jednotlivých metod jsou u popisu konkrétních metod.

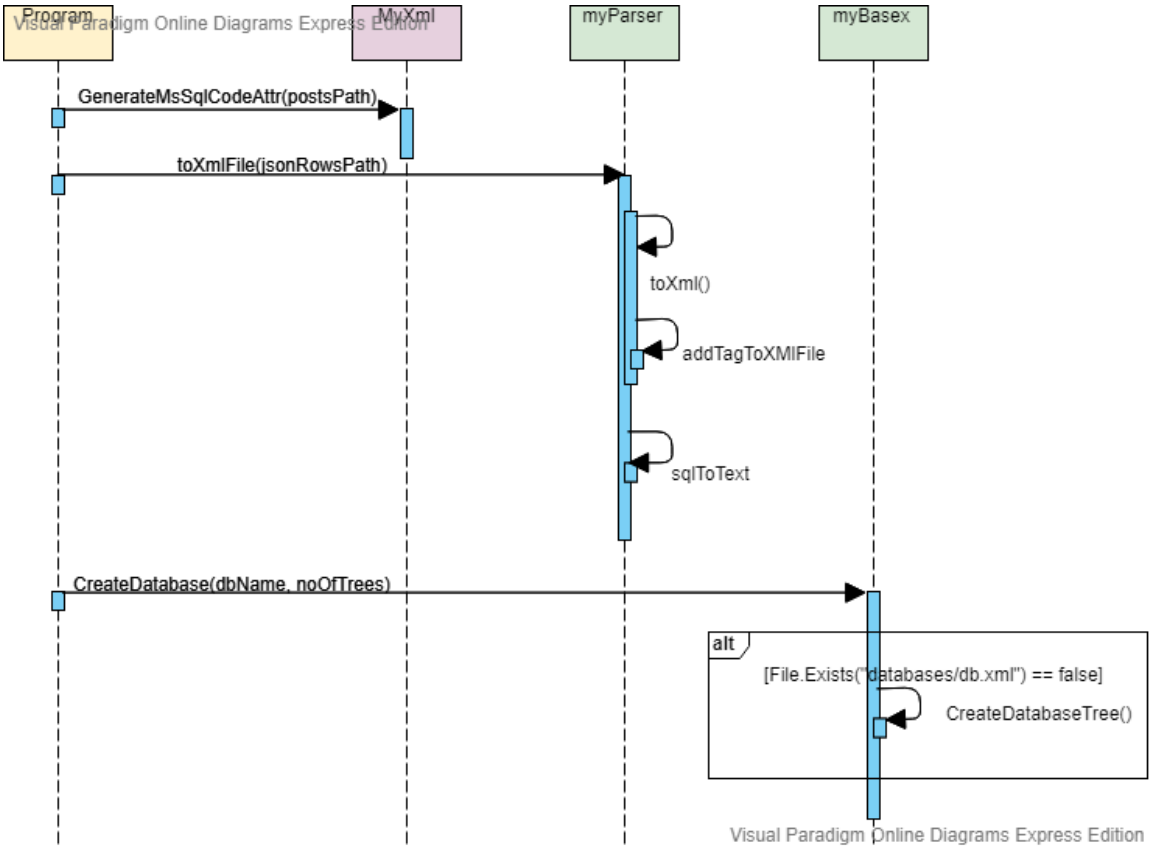
Algoritmus 1: Metoda main

input : *args*
output: -

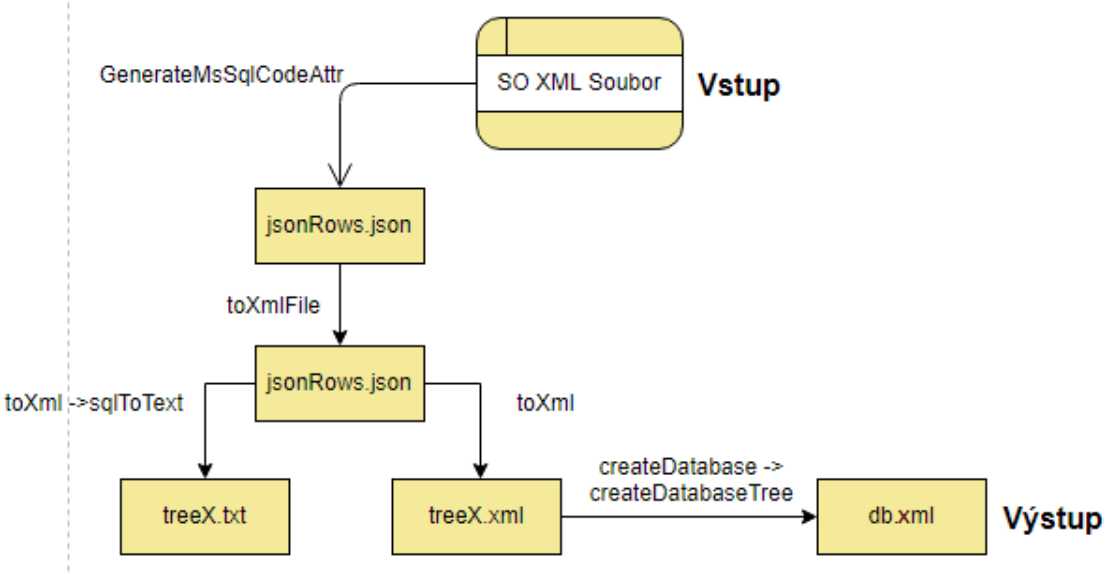
```
1 MyXml xml = new MyXml();
  /* viz algoritmus 2                                              */
2 xml.GenerateMsSqlCodeAttr(postsPath, null);
3 myParser mp = new myParser(EDbVendor.dbvmssql);
  /* viz algoritmus 3                                              */
4 mp.toXmlFile("jsonRows.txt");
5 myBasex mb = new myBasex();
6 mb.CreateDatabase("sql_analyzer",mp.noOfTrees, createDB);
```



Obrázek 8: StackOverflow Analyzer - třídní diagram



Obrázek 9: StackOverflow Analyzer - sekvenční diagram



Obrázek 10: StackOverflow Analyzer - data flow diagram

Nyní bude následovat detailnější popis jednotlivých tříd.

3.1.1.1 Třída MyXml MyXml je úplně první třída, která zpracovává XML soubor z webu StackOverflow. Třída obsahuje kromě metod, které vedou k vytvoření databáze, také pomocné metody. Jelikož má stažený XML soubor z webu StackOverflow kolem 72GB (staženo v září 2019), tak je téměř nemožné jej na běžném PC otevřít. Metoda kromě práce s XML také vytvoří složky files a files/trees, které se využívají k dalšímu zpracování.

Seznam metod pro třídu MyXml

- Pomocné metody
  - **GenerateMyXml** - tato metoda má 2 argumenty, a to cestu ke XML souboru z webu StackOveflow a počet řádků. Tato metoda slouží k vytvoření kopie SO XML souboru, s tím, že nově vzniklý soubor bude mít tolik řádků XML, kolik definujeme parametrem. Vygenerované XML slouží k nahlédnutí na strukturu XML souboru z webu StackOverflow, které je příliš velké a není možné jej na běžném PC otevřít. Metoda tedy byla nezbytná pro mou představu o struktuře XML, aby jej bylo možné zpracovat. Ve finální verzi StackOverflow Analyzer aplikace se tato metoda nepoužívá. Metoda nevrací nic.
  - **GetCodeAttr** - tato metoda má 2 parametry, a to cestu k SO XML souboru a počet řádků. Jedná se o pomocnou metodu, která extrahuje z XML obsah textu mezi tagy <code></code>. Text mezi tagy <code></code> je všechen text, který obsahuje zdrojové kódy v otázkách a odpovědích na webu StackOverflow. Metoda z XML z webu StackOverflow prozkoumá tolik řádků, kolik definujeme druhým parametrem této metody. Metoda vrací list stringů, tedy List<string>.
- Metody vedoucí ke zpracování SO XML souboru a k vytvoření BaseX databáze
  - **GenerateMsSqlCodeAttr**(viz algoritmus 2) - metoda přijímá 2 parametry, a to cestu k SO XML souboru. Druhý parametr je nepovinný a jedná se o počet řádků, kolik se má zpracovat. Tato metoda vytváří soubor jsonRows.json, který drží data v následující struktuře:
    - id - id otázky
    - parent id - pokud se jedná o odpověď, pak je zde id otázky
    - codes - pole s kódama různých dotazů (pole z toho důvodu, že někdy je kódů více v jedné otázce)
    - tag - obsahuje tagy dané otázky, pro další zpracování

    Na JSON soubor je po zpracování možno nahlédnout ve složce files/jsonRows.json. Tato metoda tedy zpracuje SO XML soubor do JSON souboru. Metoda iteruje po

řádcích SO XML souboru a postupně provádí různé operace. Nejdříve se řádek dekóduje pomocí funkce HtmlDecode - řádek 3. Následně se pomocí regexu vyberou tagy, id, parent id a kódy z řádků, které obsahují tagy zvolených databází - řádky 5-7. Nakonec se vytvoří objekt typu Row, do kterého se data nahrají a tento objekt se pak serializuje na json a zapíše do souboru - řádek 7. Pro práci s JSON souborem se používá balíček Newtonsoft.Json. Funkce jinak nic nevrací.

Algoritmus 2: Metoda GenerateMsSqlCodeAttr

input : *filename*, *rows* default is null
output: jsonRows.json

```
1 StreamWriter outputFile("jsonRows.txt");
2 foreach radek ∈ SO XML soubor do ;
3 dekódujeme zrovna procházený řádek
4 if pokud radek obsahuje SQL select příkaz a tag "sql-server", nebo "postgresql" then
5 |  int idRow = hodnota atributu id ve zpracovávaném řádku ;
6 |  int idParent = hodnota atributu parentId ve zpracovávaném řádku ;
7 |  Row row = do objektu typu Row uložíme id, parentId, tag a sql dotaz následně
  |    takový objekt převedeme do json zápisu a záznam připíšu do outputFile ;
8 end
9 write lineCounter lines to outputFile ;
```

3.1.1.2 Třída myParser O další krok ve zpracování SO XML souboru se stará třída myParser. Třída má několik metod, přičemž jedna metoda se opět ve finální verzi StackOverflow aplikace nepoužívá a sloužila jen pro statistický přehled. Hlavní úkol této třídy spočívá ve vytvoření derivačních stromů.

Seznam metod pro třídu myParser

- Pomocné metody
  - **parseFile** - metoda parseFile nám do konzole vypíše, kolik položek z jsonRows.json, který se předává jako parametr, se podařilo zparsrovat, a kolik ne. Metoda není použita ve finální verzi StackOverflow Analyzer, ale sloužila pouze jako kontrola při vývoji. Tato metoda jinak nic nevrací.
- Metody vedoucí k vytvoření BaseX databáze
  - **toXmlFile**(viz algoritmus - 3) - jedná se o hlavní metodu, která tvoří derivační stromy. Metoda má jeden parametr, a to cestu k JSON souboru jsonRows.json, který zpracovává. ToXmlFile má hlavní smyčku while, ve které po řádcích čte jsonRows.json - řádek 7. Načtený řádek deserializuje do objektu Row. Poté se cyklem projede přes codes, obsahující texty mezi tagy <code></code> - řádek 9. Tyto texty jsou nejprve ošetřeny o speciální znaky, které narušují parser od gudusoftu. Třída TGSqlParser

je z externí knihovny gudusoft.gsqlparser. V konstruktoru objektu TGSqlParser se nastaví databázový vendor, vlastnost sqltext se nastaví na ošetřený text a následně se metodou parse() zkusí text zparsrovat - řádek 11. Metoda parse vrací číslo 0, pokud parsrování proběhlo v pořádku. Pokud se parsrování povede, volá se metoda toXML a následně sqlToText. Nejdříve se podíváme na metodu toXML.

Algoritmus 3: Metoda toXmlFile

```
   input  : string fileName
   output: treeX.xml, treeX.txt
 1 read file fileName;
 2 string ln;
 3 Row jsonObj;
 4 TGSqlParser sqlparser(this.dbVendor);
 5 int fc = 0;
 6 string filePathTree;
 7 while read line of file is not null do
 8 |   try jsonObj = json to object type Row from current line;
 9 |   foreach sqlText in jsonObj.codes do
10 |   |   Provedeme odstranění escape sekvencí z sql textu ;
11 |   |   int ret = sqlparser.parse(sqlText);
12 |   |   if ret == 0 then
13 |   |   |   filePathTree = "files/trees/tree"+ fc;
   |   |   |   /* viz algoritmus 4                                          */
14 |   |   |   toXML(sqlText, this.dbVendor, filePathTree, jsonObj.tag, jsonObj.id,
   |   |   |    jsonObj.idParent, regexed);
15 |   |   |   zavoláním metody sqlToText se zapíše text sqlText do souboru filePathTree
16 |   |   end
17 |   end
18 end
```

– **toXML**(viz algoritmus **??**) - statická privátní metoda, která vytváří soubory treeX, kde X je číslo od 0 do počtu vytvořených stromů. Každý tree se skládá ze dvou souborů. Jeden soubor je derivační strom a jedná se například o tree0.xml. Druhý soubor, například tree0.txt, obsahuje sql dotaz, podle kterého byl strom sestaven. Txt soubor je ale vytvářen v jiné metodě. Ukládání do XML zde probíhá pomocí třídy xmlVisitor, která je opět z balíku gudusoftu. Po uložení do souborů metoda toXml ještě zavolá další privátní statickou metodu, a to addTagToXMlFile.

Algoritmus 4: Metoda toXml

input : String $sqlText$, EDbVendor $dbVendor$, string $fileXml$, string tag, int id, int $parent_id$, string $regexed$

output: treeX.xml s konkrétním sufixem X

```
1 TGSqlParser sqlparser = TGSqlParser(dbVendor);
2 string xmlFile = fileXml + ".xml";
3 int ret = sqlparser.parse(sqlText);
4 if ret == 0 then
5 |   xmlVisitor xv2;
6 |   xv2.run(sqlparser);
7 |   xv2.writeToFile(xmlFile);
  |   /* viz algoritmus 5                                              */
8 |   addTagToXMlFile(xmlFile, tag, id, parent_id, regexed);
9 end
```

– **addTagToXMlFile**(viz algoritmus 5) - tato metoda upravuje vygenerovaný derivační strom. Přidává do derivačního stromu tagy <tags></tags>, mezi které vkládá tagy <tag></tag>, mezi které vkládá tagy uložené v jsonRows.json. Dále metoda také přidává tag <rawquery></rawquery>, který obsahuje čistý dotaz, stejně jako treeX.txt. Jsem si vědom, že treeX.txt v podstatě pozbývá smyslu, ale jednalo se o užitečnou a rychlou kontrolu v době, kdy jsem rawquery do stromů nevkládal. Vkládání rawquery se implementovalo až později, neboť jsem došel k závěru, že by bylo dobré tyto rawquery vracet v napsaných XQuery dotazech. Tag rawquery má dva atributy, a to id a parent id, které je taktéž vhodné vracet XQuery dotazem.

Algoritmus 5: Metoda addTagToXMLFile

input : string *xmlFileName*, string *tag*, int *id*, int *parent_id*, string *sqlText*
output: upravený treeX.xml s konkrétním sufixem X

```
1  string firstLine = first line of xmlFileName file;
2  string secondLine = second line of xmlFileName file;
3  string[] tags = split tag by »<";
4  String xmlTagList = "";
5  String clearedT;
6  foreach t in tags do
7  |  clearedT = odstraníme znaky «"a »"z proměnné t;
8  |  xmlTagList += <tag> + clearedT + <tag> + new line;
9  end
10 tag = <rawquery id="id"parent-id="parent_id">sqlText</rawquery> + new line +
     <tags> + new line + xmlTagList + </tags>;
11 var lines = read all lines of xmlFileName file;
12 write all lines excluding first two to the xmlFileName file;
13 string currentContent;
14 if File xmlFileName Exists then
15 |  currentContent = read all text of xmlFileName file;
16 end
17 write following text to the xmlFileName file: firstLine + New Line + secondLine +
     New Line + tag + New Line + currentContent;
```

– **sqlToText** - stará se o ukládání treeX.txt.

Nyní jsme tedy ve stavu, kdy jsou v adresáři files/trees vytvořeny derivační stromy. Zde práce třídy myParser končí a na řadu přichází třída myBasex.

3.1.1.3 Třída myBasex Třída myBasex se stará o poslední krok ve vytváření databází. Stará se vytvoření databáze a ve webové části této aplikace zajišťuje komunikaci s databází.

Seznam metod pro třídu myParser

- **CreateDatabase**(viz algoritmus **??**) - jedná se o hlavní metodu, která má za úkol vytvořit databázi. Metoda nejdříve testuje, zda existuje XML soubor, ze kterého se databáze vytvoří. Pokud tento soubor neexistuje, tak jej pomocí metody CreateDatabaseTree vytvoří. Pokud soubor pro vytvoření databáze existuje, vytvoří se instance třídy Session, která na portu 1984 komunikuje s BaseX serverem. Pomocí takto vytvořené instance se příkazem

```
CREATE DB nazev_databaze cesta_k_xml_souboru
```

Výpis 9: Vytvoření BaseX Databáze

vytvoří databáze. V metodě se ještě nastaví počet zparsrovaných stromů, který do metody předává parametrem třída myParser, která tuto informaci uchovává. Nakonec se uzavře

spojení pomocí close metody aplikované na instanci třídy Session. Metoda vrací boolean, podle toho, zda se databáze povede vytvořit, nebo ne.

- **CreateDatabaseTree**(viz algoritmus 6) - jedná se o metodu, která vytváří XML soubor, pomocí něhož je pak vytvořena celá BaseX databáze. BaseX považuje jeden XML soubor jako jednu databázi. Jelikož chceme mít databázi v této aplikaci jednu, je třeba stromy pospojovat a obalit. Konkrétně se tedy do nově vznikajícího XML nejprve nahraje definiční XML tag s verzí xml, kódováním a informací, zda se jedná o standalone. Dále se do souboru připíše otevírací tag <trees>. Následně se cyklem projíždí a zpracovávají derivační stromy. Z XML stromů se odstraní XML namespace a dekódují se znaky, které by ve zpracování mohly dělat problémy. Následně se takto upravený strom připíše do XML souboru databáze. Nakonec je do XML souboru databáze dopsán zavírací tag </trees>. Pokud vše proběhne v pořádku, metoda vrátí true, pokud ne, vrátí false.

Algoritmus 6: Metoda CreateDatabaseTree

input : -
output: db.xml

```
1  int i = 0;
2  string treePath = "trees/tree"+ i + ".xml";
3  string contents;
4  writetext = soubor db.xml;
5  do souboru writetext zapíšeme následující text: <?xml
     version="1.0"encoding="utf-8"standalone="no"?>;
6  do souboru writetext zapíšeme následující text: <trees>;
7  while dokud je i menší, nebo rovno počtu souborů treeX.xml do
8  |  contents = obsah souboru treePath;
9  |  z proměnné contents odstraním následující tři řádky:;
10 |  <?xml version="1.0"encoding="utf-8"standalone="no"?>;
11 |  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance;
12 |  xmlns="http://www.sqlparser.com/xml/sqlschema/1.0;
13 |  do souboru writetext zapíšeme následující text: <tree> + contents + </tree>;
14 |  i++;
15 |  treePath = "trees/tree"+ i + ".xml";
16 end
17 do souboru writetext zapíšeme následující text: </trees> ;
```

V popsaných metodách se vyskytovala třída Session. Tato třída je z namespace BaseXClient, z čehož je patrné, že se jedná o nabízený způsob komunikace s databází v C# přímo BaseX systémem. Co se týče zpracování dat a vytvoření databáze, tak je to vše. Nicméně ještě popíši poslední třídu, která se ve finální verzi aplikace sice nepoužívá, ale při vývoji se používala.

3.2 StackOverflow Analyzer - webová aplikace

Webová aplikace slouží primárně pro psaní a spouštění XQuery dotazů. Je napsána v C#, s použitím ASP.NET MVC. Nicméně databázovou část zde obsluhuje StackOverflow Analyzer - konzolová část aplikace, namísto MVC modelu. O chod celé aplikace se stará HomeController, který obsahuje následující metody:

- **Index** - metoda pro index stránku, kterou ve finální verzi projektu nepoužívám
- **loadOptions** - Tato metoda se stará o načítání předdefinovaných XQuery dotazů. Všechny předdefinované dotazy jsou uloženy ve složce queries ve složce s projektem. Metoda obsahuje smyčku, ve které ve složce queries prochází všechny soubory s příponou .json. Načtené hodnoty dosadí do html tagů <option></option>, které pak tato metoda jako string vrací.
- **DoQuery** - query je hlavní metoda pro zasílání dotazů na databázi a vracení příslušných výsledků. Metoda přijímá dva parametry, a to samotný dotaz a název databáze. Název databáze je nepovinný, pokud není vyplněn, nastaví se podle názvu, který je uložen v třídní proměnné - sql_analyzer. Jelikož aplikace nevyužívá vlákna a nemůže být tedy nonstop připojena k jedné session instanci, musíme pomocí nové instance třídy Session vytvořit nové připojení. Následně se pomocí dotazu

```
OPEN nazev_databaze
```

Výpis 10: Připojení k BaseX Databázi

otevře spojení k databázi. Tento dotaz se posílá pomocí metody Execute, která patří třídě Session. První parametr metody Execute je příkaz pro databázi a druhý parametr je stream, kam se vypíše výsledek. V C# je možnost použít MemoryStream. MemoryStream v aplikaci tedy nastavím jako druhý parametr metody Execute a pokud na tento stream zavolám metodu ToArray() a na její výsledek aplikuji metodu Encoding.ASCII.GetString, tak se extrahuje text v čitelné podobě do proměnné typu string. Tento string potom metoda vrací. Samozřejmě se nesmí zapomenout zavřít spojení pomocí Session.Close(). Stránkování je řešeno tak, že se výsledky ukládají do souboru na server a dle čísla stránky se načtou požadovaná data. V metodě se dále volá metoda loadOptions, která nahraje roletky options. Pomocí ViewBag se výsedek dotazu i string obsahující <option></option> tagy nastaví k zobrazení ve view, které je touto metodou také vráceno. Konkrétně se jedná o view Results.
- **Info** - metoda Info zasílá na databázi příkaz info. Vrácen je string, který databáze vrátí. String obsahuje nejrůznější informace o momentálně otevřené databázi. Tato metoda se využívá ve webové části StackOverflow Analyzer aplikace k ověření, zda funguje připojení k databázi, popřípadě k výpisu informací o databázi.

- **GetPredefined** - metoda, která se pomocí jQuery post funkce vyvolá z view. Přijímá jeden parametr, a to id. V metodě je cyklus, který projíždí předdefinované XQuery dotazy a porovnává jejich id s id z parametru. Pokud se id rovnají, tak vrátí string obsahující XQuery dotaz. Pokud se nenajde záznam s odpovídajícím id, tak se vrátí hláška, která o tom uživatele informuje.
- **Error** - metoda, která se volá v případě, že je napsaná URL, která neexistuje. Jedná se o defaultní Error metodu MVC aplikace.

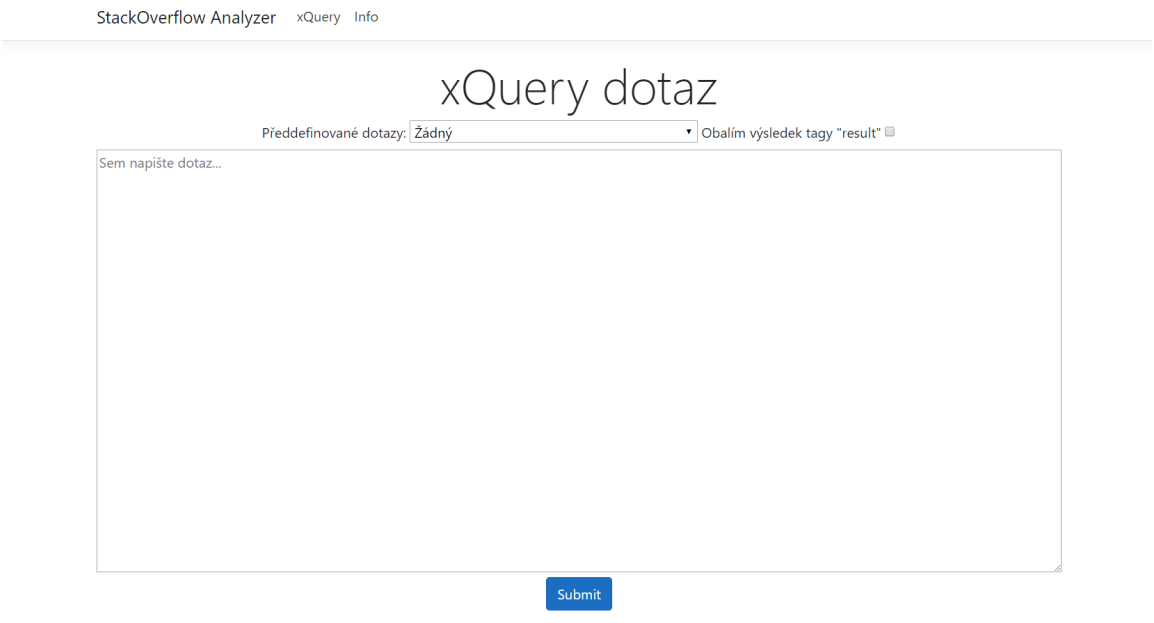
Jak jsem již v metodě loadOptions nastínil, kromě vypsání vlastního XQuery dotazu aplikace obsahuje i několik předdefinovaných dotazů. Dotazy je možné vložit i vlastní k budoucímu použití. Dotazy jsou uloženy ve formátu json ve složce queries, která je ve složce s projektem. Formát json souboru je následující:

- id - číselné id, je možno nastait jakékoliv volné číslo, doporučiji však číslovat postupně
- name - je to text, který se zobrazí v selectu mezi tagy <option></option>
- query - samotný XQuery dotaz

Takto nadefinovaný dotaz se pak automaticky zobrazí v select roletce na webu.

3.2.1 views

Aplikace obsahuje celkem tři hlavní a jedno vedlejší Views. Index - zobrazí se na základní stránce a obsahuje navigaci, formulář pro psaní XQuery dotazu. Formulář obsahuje select roletku s předdefinovanými XQuery dotazy a textareu pro psaní vlastního XQuery dotazu - obrázek 11. Po odeslání tohoto formuláře je nám zobrazeno druhé view, a to Results. View Results obsahuje taktéž navigaci a formulář z view Index. Navíc je zde ale textarea, která obsahuje výsledek napsaného dotazu. Textarea je v módu readonly. Třetí hlavní view je Info. View obsahuje informace o databázi a zároveň může sloužit jako rychlý test, zda databázové připojení v pořádku funguje. Poslední view, které aplikace obsahuje, je view Error. Toto view je vyvoláno, pokud je zadána neznámá URL adresa.



Obrázek 11: StackOverflow Analyzer - Webová část aplikace - formulář

4 Testování na vlastních datech

Tato kapitola se věnuje testování aplikace. Zhodnotí se zde, které kroky procesu jsou nejnáročnější. Zobrazí se jejich časová náročnost. Předmětem měření jsou jednotlivé metody, které se podílejí na zpracování SO XML souboru a vytvoření databáze. Měření se provádělo se vstupním SO XML Souborem, který v době stažení zabíral 72 833 326 KB.

PC sestava, na které probíhalo měření:

- **CPU** - Intel Core i5-4590
- **RAM** - HyperX Fury Black 8GB (2x4GB) DDR3 1866
- **Základní deska** - GIGABYTE GA-Z97X-Gaming 3 - Intel Z97
- **Operační systém** - Windows 10 Education - 64bit

Pořadí	Metoda	Rychlost[ms]	Rychlost [h] (zaokrouhleno)	% z celkového běhu aplikace
1	GenerateMsSqlCodeAttr	8577297	2.38	73%
2	toXmlFile	1585218	0.44	14%
3	CreateDatabase	859795	0.24	7%
4	CreateDatabaseTree	737440	0.2	6%
Celkem		11759750	3.27	100%

Tabulka 2: Výsledky měření rychlostí jednotlivých metod StackOverflow Analyzer - konzolové části

Z naměřených hodnot v tabulce 2 je vidět, že nejdéle běžící metoda je GenerateMsSqlCodeAttr. Je to dáno tím, že tato metoda prochází celý SO XML soubor a po řádcích jej zpracovává. Vytváří se zde soubor jsonRows.json, ve kterém jsou již pouze relevantní řádky, a to ty, které mají tag sql-server, nebo postgresql a obsahují dotaz SELECT.

Na druhé pozici vidíme toXmlFile. Tato metoda běží už jen 14% času z běhu aplikace, protože zpracovává řádky souboru jsonRows.json, jehož velikost je oproti SO XML souboru podstatně menší. Soubor jsonRows.json zabíral 368MB. Nicméně na druhém místě v tabulce je proto, že metoda zajišťuje vytvoření treeX.xml a treeX.txt. Takovýchto dvojic treeX.txt a treeX.xml se vygenerovalo 176 441.

Na třetí pozici je metoda CreateDatabase. Metoda běží 7%, ale s tím, že se v metodě spouští CreateDatabaseTree, která běží 6% z celkového času běhu aplikace. Takže většina času se využívá právě na vytvoření souboru db.xml. Metoda sama o sobě pouze zajistíšťuje případné volání metody CreateDatabaseTree a zasílá dotaz na BaseX server pro vytvoření databáze s užitím db.xml souboru. Soubor db.xml zabíral 1,29GB.

Pořadí	XQuery dotaz	Rychlost[ms]	Nalezených výsledků
1	Zastaralé propojení tabulek pomocí WHERE	9263.75	3859
2	Obsahující nezávislý poddotaz v EXISTS	3968..76	105
3	Obsahující tabulku, která není propojená s ostatními	2494.67	2852
4	StartFragmentRedundantní konstrukce distinct	181.87	0

Tabulka 3: Výsledky měření rychlostí jednotlivých XQuery dotazů

V tabulce 3 můžeme vidět časy běhu jednotlivých XQuery dotazů. XQuery dotazy jsou v příloze této práce (D-G). Z výsledků vyplývá, že nejčastější chybou je propojení tabulek pomocí WHERE. Je to celkem pochopitelné, protože se v minulosti toto propojení běžně používalo. XQuery dotaz funguje tak, že postupně prochází tabulky z klauzule FROM a pomocí atributu type lze poznat jakým způsobem je tabulka připojena. Pokud dotaz obsahuje více než jednu tabulku ve FROM, která není připojená pomocí JOIN, tak je tento dotaz vypsán. Zajímavé také je, že dotaz hledající redundantní použití DISTINCT v poddotaze exists a IN (a jejich negacích), nenašel jediný výsledek.

5 Závěr

Výsledkem mé práce jsou dvě aplikace, které dohromady mohou sloužit jako prostředek k zanalyzování dotazů na webu StackOverflow. Data jsou na webu k dispozici v surové podobě, ve které s nimi není možné pracovat. Aplikace vytvořená v rámci této bakalářské práce slouží jako nástroj pro práci s těmito daty, což ocení především lidé, kteří se zabývají psaním optimalizátorů. Z dat mohou zjistit jaké konstrukce uživatelé databází reálně používají, a jakých chyb se dopouští. Konzolová část StackOverflow aplikace se stará o zpracování SO XML souboru. Webová aplikace potom slouží ke psaní a spouštění XQuery dotazů.

Při spuštění konzolové části aplikace se ze zadaného SO XML souboru vytváří soubory, které vedou až k vytvoření finálního souboru db.xml. Popřípadě tato aplikace i vytvoří BaseX databázi na lokálním serveru. Jelikož vstupní SO XML soubor zabírá kolem 80GB, aplikace běží poměrně dlouho. Viz kapitola Testování na vlastních datech.

Další částí této bakalářské práce je webová část aplikace, která pak s touto databází pracuje. Po spuštění vidíme formulář, do kterého můžeme napsat vlastní XQuery dotaz, nebo můžeme zvolit jeden z předpřipravených dotazů pro demonstraci. Takto zapsaný dotaz se zašle na databázi a uživateli jsou vypsány výsledky. Jelikož se předpokládá, že výsledků bude mnoho, je v aplikaci také stránkování. Kromě toho nabízí webová aplikace ještě informace o databázi, což může sloužit i jako ověření, že je aplikace úspěšně připojena k databázi.

Tato bakalářská práce mě naučila mnoho nového. Například jsem se seznámil s databázovým systémem BaseX, nebo s technologií XQuery, přičemž obě tyto technologie jsou používány i mimo akademickou půdu. Dále jsem si prohloubil znalosti jazyka C#, a to jak v použití na implementaci konzolové aplikace, tak té webové, postavené na technologii ASP.

Zadání práce se podařilo splnit.

Díky této práci jsem objevil další, velmi zajímavé zákoutí IT průmyslu.

Literatura

1. SQLPARSER, Gudusoft. *Why do people choose to use our powerful SQL Parser?* Version 1. Dostupné také z: http://www.sqlparser.com/.
2. SQLPARSER, Gudusoft. *Supported Databases.* Dostupné také z: http://www.sqlparser.com/sql-parser-supported-databases.php.
3. HORDĚJČUK, Vojtěch. *Formální gramatika.* Verze 1. Dostupné také z: http://voho.eu/wiki/formalni-gramatika/.
4. DOC. RNDR PETR JANČAR, CSc. *Úvod do teoretické informatiky – učební text.* Reading, Mass.: Ediční středisko VŠB-TUO, 2007.
5. SQLPARSER, Gudusoft. *SQL Parser How-To.* Dostupné také z: http://www.sqlparser.com/sql-parser-how-to.php.
6. *Workflow parsing* [online] [cit. 2019-04-24]. Dostupné z: http://www.sqlparser.com/images/workflow.gif.
7. SQLPARSER, Gudusoft. *Decoding SQL grammar — select statement.* Dostupné také z: http://www.dpriver.com/blog/list-of-demos-illustrate-how-to-use-general-sql-parser/decoding-sql-grammar-select-statement/.
8. *XQUERY COVER PAGE.* Dostupné také z: https://www.w3.org/TR/xquery/all/.
9. *XPath Nodes.* Dostupné také z: https://www.w3schools.com/xml/xpath_nodes.asp.
10. *XPath - Osy* [online] [cit. 2014]. Dostupné z: https://www.kosek.cz/xml/xslt/xpath.html.
11. *XQuery Syntax.* Dostupné také z: https://www.w3schools.com/xml/xquery_syntax.asp.
12. *XQuery Adding Elements and Attributes.* Dostupné také z: https://www.w3schools.com/xml/xquery_add.asp.

A Manuál

V přiložených souborech k této práci je manuál, popisující spuštění aplikace.

B StackOverflow Analyzer - konzolová část aplikace

V přiložených souborech k této práci je StackOverflow Analyzer - konzolová část aplikace. Obsahuje všechny zdrojové kódy a soubor solution pro otevření aplikace ve Visual Studiu.

C StackOverflow Analyzer - webová část aplikace

V přiložených souborech k této práci je StackOverflow Analyzer - webová část aplikace. Obsahuje všechny zdrojové kódy a soubor solution pro otevření aplikace ve Visual Studiu.

D XQuery dotaz - Používající zastaralého zápisu spojení tabulek.

```
for $r in /trees/tree//select_statement
let $from_clause := $r/query_expression/query_specification/from_clause/
    table_reference[@type="objectname"]
where count($from_clause) > 1
return
<result>
id: {data($r/../preceding-sibling::rawquery/@id)}
<br />
parent-id: {data($r/../preceding-sibling::rawquery/@parent-id)}
<br />
{$r/../preceding-sibling::rawquery/text()}
<br />
</result>
```

Výpis 11: XQuery dotaz - Používající zastaralého zápisu spojení tabulek.

E XQuery dotaz - Obsahující nezávislý poddotaz v EXISTS.

```
for $stmt in /trees/tree//select_statement

let $outer_table := ($stmt/query_expression/query_specification/from_clause/
    table_reference[not(.//alias_clause)]//table_name/object_name/text() union
 $stmt/query_expression/query_specification/from_clause/table_reference//
     alias_clause//full_name/text() union
$stmt/query_expression/query_specification/from_clause/named_table_reference[
    not(.//alias_clause)]//table_name/object_name/text() union
$stmt/query_expression/query_specification/from_clause/named_table_reference//
    alias_clause//full_name/text())

let $inner_table := ($stmt/query_expression/query_specification/where_clause//
    exists_expr//where_clause//column_referenced_expr//object_name/text())
where not($outer_table = $inner_table) and $inner_table
return
<result>
id: {data($stmt/../preceding-sibling::rawquery/@id)}
<br />
parent-id: {data($stmt/../preceding-sibling::rawquery/@parent-id)}
<br />
{$stmt/../preceding-sibling::rawquery/text()}
<br />
</result>
```

Výpis 12: XQuery dotaz - Obsahující nezávislý poddotaz v EXISTS

F XQuery dotaz - Obsahující tabulku, která není propojena, žádným spojením s ostatními.

```
for $r in /trees/tree//select_statement
let $outer_table := ($r/query_expression/query_specification/from_clause/
    table_reference[not(.//alias_clause)]//table_name/object_name/text() union
$r/query_expression/query_specification/from_clause/table_reference//
    alias_clause//full_name/text() union
$r/query_expression/query_specification/from_clause/named_table_reference[not(.
    //alias_clause)]//table_name/object_name/text() union
$r/query_expression/query_specification/from_clause/named_table_reference//
    alias_clause//full_name/text())

let $from_count := $r/query_expression/query_specification/from_clause/
    table_reference

let $where_tables := ($r/query_expression/query_specification/where_clause/
    condition/*/*/column_referenced_expr//object_name/text() union
$r/query_expression/query_specification/where_clause/condition/*/*/*/*/
    column_referenced_expr//object_name/text())

where (not((every $outer_table in $outer_table satisfies $where_tables[contains
    (.,$outer_table)])) and count($from_count) > 1)

return
<result>
id: {data($r/parent::statement/preceding-sibling::rawquery/@id)}
<br />
parent-id: {data($r/parent::statement/preceding-sibling::rawquery/@parent-id)}
<br />
{$r/parent::statement/preceding-sibling::rawquery/text()}
<br />
</result>
```

Výpis 13: XQuery dotaz - Obsahující tabulku, která není propojena, žádným spojením s ostatními

G XQuery dotaz - Redundantní konstrukce distinct (kupříkladu v poddotazu IN).

```
for $r in /trees/tree//select_statement
let $distinct := ($r/query_expression/query_specification/where_clause//in_expr
    //distinct_clause,
$r/query_expression/query_specification/where_clause//exists_expr//
    distinct_clause)
where (count($distinct) >= 1)
return
<result>
id: {data($r/../preceding-sibling::rawquery/@id)}
parent-id: {data($r/../preceding-sibling::rawquery/@parent-id)}
{$r/../preceding-sibling::rawquery/text()}
</result>
```

Výpis 14: XQuery dotaz - Redundantní konstrukce distinct (kupříkladu v poddotazu IN